PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/46, 9/40

A1

(11) International Publication Number: WO 99/10807

(43) International Publication Date: 4 March 1999 (04.03.99)

(21) International Application Number: PCT/US98/15022

(22) International Filing Date: 20 July 1998 (20.07.98)

(30) Priority Data:

60/056,634	22 August 1997 (22.08.97)	US
08/984,167	3 December 1997 (03.12.97)	US

(71) Applicant: CIRRUS LOGIC, INC. [US/US]; 3100 West Warren Avenue, M/S 521, Fremont, CA 94538-6419 (US).

(72) Inventor: DEANS, Scott; 7309 Berkshire Downs Drive, Raleigh, NC 27604 (US).

(74) Agent: SHAW, Steven; Cirrus Logic, Inc., 3100 West Warren Avenue, M/S 521, Fremont, CA 94538-6419 (US).

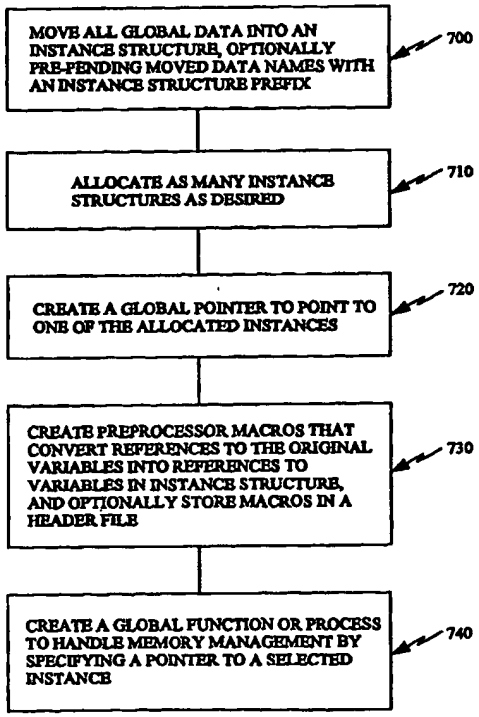
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HU, ID, IL, IS, JP, KE, KG, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: METHOD AND SYSTEMS FOR CREATING MULTI-INSTANCED SOFTWARE WITH A PREPROCESSOR

(57) Abstract

A preprocessor is utilized to create multi-instantiated code from single instance code so that multiple copies of the single instance code can be run without changing the source code of the single instance code. This document describes a simple method of converting ordinary single-instance software into software that supports multiple instances, each with its own data, running simultaneously on a system. This method enables multiple instance support without modifying the original functions in the source code.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

METHOD AND SYSTEMS FOR CREATING MULTI-INSTANCED SOFTWARE WITH A PREPROCESSOR

CROSS-REFERENCES TO RELATED APPLICATIONS

This application claims priority from provisional patent application Serial No. 60/056,634 filed August 22, 1997, entitled CREATING MULTI-INSTANCED SOFTWARE EFFORTLESSLY WITH THE PREPROCESSOR which is incorporated herein by reference in its entirety.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to computer technology and, more particularly, to creating multiple-instance software from single-instance source code utilizing a compiler's preprocessor.

2. Description of Related Art

Most software is written to use a "single instance", which means that it is designed to run only one copy of the program and its data at a given time on a single system. Some other software can be shared by more than one "client" program at a time, allowing each client to run a copy of the program with its own set of data. This software can be called "multi-instanced", in that more than one "instance" of the code can be running at one time.

A good example of multi-instanced software is that of a Windows DDL (Dynamically-Linked Library), which can be shared among many programs at once. A Dynamically-Linked Library (or DLL) is a collection of program functions and data in the Microsoft Windows software architecture. On its own, a DLL performs no useful function, as it is merely a library of code that is linked together into one single unit, or file, on a computer's magnetic storage medium. Client applications running on a Windows system can dynamically load these libraries into computer memory and access the program functions contained therein. A DLL uses the client application's data space to hold its data, so the DLL is able to keep each client program's data separate from the data of other client programs. That approach, however ,requires operating system support to manipulate the data segments and keep track of which process owns the data. It also requires each client's copy of the DLL's data to exist in a different "data segment". A data segment is a section of computer memory that is reserved by the operating system for exclusive use by a client program. Intel architecture microprocessors provide support for manipulating data segments via the DS hardware register.

In the prior art, when one wished to change software from single instance to handle multiple instances, a complete redesign of the software was required. This caused many problems. First, significant training was required of customer personnel in order to understand and utilize the redesigned software. Further, it was expensive to maintain because maintenance personnel would have to be conversant with two versions of the software, one a single instance version and one a multiple instance version. Further, when running the software, plural copies of the code would typically run, each having its own data which produced a substantially redundant situation. Thus, it would be desirable if the same code could be used for single-instance and multi-instance implementations. Further, it would be desirable if only one copy of the code were running which would service all of the instances needed.

SUMMARY OF THE INVENTION

In accordance with the invention, single-instance software is converted into software that supports multiple instances, each with its own data, running simultaneously on a system. The approach enables multiple instance support without modifying the original source code. This is preferably done at compile time using the preprocessor.

Also in accordance with the invention, modems or computers run a multi-instantiated modem code set(s) or portions of a modem code set, such as a protocol stack or data link control element.

Also in accordance with the invention, a processor having no memory management unit is enabled to operate as if it had one.

Also in accordance with the invention, a single task operating system, such as DOS, is enabled to run as a multi-task system.

When multiple instances of the same piece of code are running, only one copy of the code need be resident in memory. However, by changing instances, one can create a global change in behavior of the currently executing software without changing the source code.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1A is a view of exemplary computer system suitable for use in carrying out the invention.

Figure 1B is a block diagram of an exemplary hardware configuration of the computer of Figure 1.

Figure 1C is an illustration of an exemplary memory medium suitable for storing program and data information in accordance with the invention.

Figure 1D is a block diagram of a network architecture suitable for carrying data and programs in accordance with some aspects of the invention.

Figure 2 illustrates an exemplary software architecture suitable for use in carrying out the invention.

Figure 3 is a block diagram of exemplary modem software suitable for use in carrying out the invention.

Figure 4 is a block diagram of a modification of exemplary software of Figure 3 for carrying out an exemplary implementation of Digital Simultaneous Voice and Data (DSVD (V.70)).

Figure 5 is a block diagram showing more details of the DSVD (V.70) layer of Figure 4.

Figure 6 is a block diagram of processing typically undertaken to compile, link, load and run source code.

Figure 7 is a flow chart of a process for creating multi-instanced software from single instance source code using the preprocessor.

Figure 8 is a flow chart of a process for using multi-instanced software to implement a DLC layer of a controllerless modem using DSVD (V.70).

Figure 9 is a diagram showing a multi-instanced DLC layer showing multiple instances of a line access protocol (LAPM).

Figure 10 is a diagram showing how multi-instanced software can be utilized to implement a plurality of protocol stacks which can be used simultaneously or independently in accordance with one embodiment of the invention.

Figure 11 is a diagram showing how multi-instanced software can be utilized to implement a plurality of modems which can be used simultaneously or independently in accordance with one embodiment of the invention.

Figure 12 is an illustration of a technique for enabling multi-tasking on a processor without operating system support and without a memory management unit in accordance with one embodiment of the invention.

Figure 13 is an illustration of a technique for enabling multi-tasking on a processor running a single task operating system.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Figure 1A illustrates a computer 100 of a type suitable for carrying out the invention. Viewed externally in Figure 1A, a computer system has a central processing unit 100 having disk drives 110A and 110B. Disk drive indications 110A and 110B are merely symbolic of a number of disk drives which might be accommodated by the computer system. Typically, these would include a floppy disk drive such as 110A, a hard disk drive (not shown externally) and a CD ROM drive

indicated by slot 110B. The number and type of drives varies, typically, with different computer configurations. The computer has the display 120 upon which information is displayed. A keyboard 130 and a mouse 140 are typically also available as input devices. This computer is also audio equipped with speakers 178 and a microphone 179.

Figure 1B illustrates a block diagram of the internal hardware of the computer of Figure 1A. A bus 150 serves as the main information highway interconnecting the other components of the computer. CPU 155 is the central processing unit of the system, performing calculations and logic operations required to execute programs. Read only memory (160) and random access memory (165) constitute the main memory of the computer. Disk controller 170 interfaces one or more disk drives to the system bus 150. These disk drives may be floppy disk drives, such as 173, internal or external hard drives, such as 172, or CD ROM or DVD (Digital Video Disks) drives such as 171. A display interface 125 interfaces a display 120 and permits information from the bus to be viewed on the display. Communications with external devices can occur over communications port 175. Computer 100 also has an audio input/output interface 177 for audio communications and a digital signal processor 176 which provides a hardware interface to a communications line and which can function as an output interface for a modem.

Figure 1C illustrates an exemplary memory medium which can be used with drives such as 173 in Figure 1B or 110A in Figure 1A. Typically, memory media such as a floppy disk, or a CD ROM, or a Digital Video Disk will contain program and data information as described more fully below for controlling the computer to enable the computer to perform its multi-instanced functions in accordance with the invention.

Figure 1D is a block diagram of a network architecture suitable for carrying data and programs in accordance with some aspects of the invention. A network 190 serves to connect a user computer 100 with one or more of a plurality of servers such as server 195. Servers provide various services, such as the download of program and data information. Users such as user 100', may also connect to the network by way of a network service provider such as ISP 180. A server such as 195 can download program and data information needed for carrying out the invention to users 100 over network 190.

Figure 2 illustrates an exemplary software architecture suitable for use in developing multi-instanced software in accordance with the invention. Typically, computer 100 has an operating system 200. In the example shown, it also has code for running a modem, in this case, a software modem 210. Applications 220 utilize the services of the operating system and the modem code in carrying out their

functionality. Modem code 210 may utilize services of the operating system in order to access external devices or may bypass the operating system and access external devices directly. This will depend upon the implementation. Accordingly, the dashed line indicates that the modem code utilizes the services of the operating system or, when it does not, the dash line is treated as not there, indicating the modem code bypasses the operating system. It is possible for the modem to partially bypass and partially utilize services of the operating system, depending upon a particular implementation.

Figure 3 is a block diagram of exemplary modem software for multi-instanced implementation in accordance with the invention. The applications interface with a DTE interface layer 300 which manages the interaction with the computer. A protocol layer 310 receives information from the data interface layer and formats it in accordance with a particular protocol in use. An optional compression layer 320 may be utilized to compress the data prior to passing it to the line I/O layer 330 which interfaces the higher layers with a digital signal processor 176 which serves as the hardware interface to a communications line. Note that in the examples shown, the software layers 300, 310, 320 and 330 together constitute a software modem and are run, in a preferred embodiment, on the host. In some embodiments, there are two software modules that interface with the DSP hardware. The Modem Control Layer, or "modem task", manipulates DSP modes. creates and destroys modem connections, and monitors modem activity. The Line I/O Layer is responsible for transferring data to and from the DSP. Alternatively, the software shown in Figure 3 can be run on a modem having a dedicated processor. Modems of the latter type are found, typically, in internal modem line cards in computer devices and as stand-alone modems which connect externally to a computer.

Figure 4 is a block diagram of a modification of exemplary software of Figure 3 for carrying out an exemplary implementation of Digital Simultaneous Voice and Data (DSVD (V.70)). When a modem is configured to handle DSVD in accordance with the recommendation V.70, there are two source interfaces. The first is a DTE layer 300 which receives data from an application before transmission over a communications line. An audio interface layer 310 receives audio from an application for transmission over the communication line as well. Recommendation V.70 specifies standards for the implementation of this functionality. However, implementations are left to the individual vendors. The DSVD layer 320 manages the audio and data information as it is transmitted over a communications link. This is described in more detail hereinafter.

Figure 5 is a block diagram showing more details of the DSVD (V.70) layer of Figure 4. There are two entities specified in the DSVD (V.70) Recommendation.

One is a control entity 500 and the second is a data link control layer (DLC) 510. The control entity 500 manages the input/output from the data interface layer and the audio interface layer. It also passes information over the select line to the data link control layer indicating the source of the information arriving. The data link control layer manages data formatting using, in this case, link access protocol, LAPM. Since the line access protocol is substantially the same for each source, one needs to have two processes running, one for the audio and one for the data. If the data link control layer were implemented, using techniques of the prior art, there would be two separate versions of the LAPM code running, one for audio and one for data. However, in accordance with one aspect of the invention, what would otherwise be single instance LAPM code is converted to multi instance software with several attendant advantages. The advantages include having only one version of the code running while being able to simultaneously serve the separate needs of the audio and data sources. How this is done is explained in more detail hereinafter. In some implementations there may be multiple control entries with the selection of data to be sent being handled by the data link control layer.

Figure 6 is a block diagram of processing typically undertaken to compile, link, load and run source code. Source code, shown symbolically at block 600 is run through a preprocessor 610 in the normal course of compiling an application. Once it has been preprocessed, the preprocessor output is applied to compiler 620 resulting in object code. The object code from the compiler is then linked (630) with header files, libraries, functions and macros to produce the final executable code which can be loaded and run as shown at 640.

Figure 7 is a flow chart of a process for creating multi instanced software from single instance source code using the preprocessor. The single instance source code which one desires to convert to multi-instance source code is managed as follows: First, one moves all global data from the source code into an instance structure, optionally prepending moved data names with an instance structure prefix (700). Essentially, one creates a template data structure which serves as a model for a plurality of instance data structures. As many instance data structures as desired are allocated (700) and a global pointer is maintained to point to one of the allocated instances (720). The global pointer is used, when running an application, to specify the particular one of the allocated instances which is to be active for processing at that point in time.

One or more preprocessor macros are created to convert references to the original variables into references to variables in the instance structure (730). These macros can be conventionally stored in a header file. Thus, when preprocessing source code, references to the original variables are converted into references to the

variables in a particular instance structure. A global function or macro is created to handle memory management by specifying a pointer to a selected or active instance (740).

This is done in the preprocessor 610 prior to compilation at 620.

An example may help clarify the operation of the process shown in Figure 7. Consider the following program.

Before

```
/* Sample program before applying invention techniques */
#include <stdio.h>

/* Function prototype */
void ShowInfo(void);

/* Global data */
int age;
char name[64];


/* Target function */
void ShowInfo (void)
{
  printf ("My name is %s.\n", name);
  printf ("My age is %d.\n", age);
}


/*
 * Main program code.
 *
 * This simple program will repeat the call to ShowInfo()
 * until the program is terminated.
 */
main ()
{
  while (1)
  {
    ShowInfo();
  }
}
```

In the example above there are only two variables, namely name and age. "Age" is a global integer and "name" is a global string. The program does a loop by which the sentences "My name is _____. My age is _____." are repeated continuously. Although this is not a useful program, it serves to illustrate in a simple fashion the power of the approach described in conjunction with Figure 7.

When the program is modified in the preprocessor in accordance with the invention. all data is placed in a data structure _"person."

The following "after" program results. This "after" program has had coniments added referring to Figure 7 of the drawings which explains what portion of the code results from the processing steps.

After

```
/* Sample program using invention techniques */
#include <stdio.h>

/* Function prototype */
void ShowInfo(void);


/* STEP 1: Move all global data into a new structure
      (called the instance structure). Prepend the original data items with the 'i_'
      prefix (or ANY other prefix desired) to help distinguish the items as instance
      data items.

      Prepending the data items is not required and is included simply to help
      distinguish instance data from other global data.
 */

/* Structure definition */
typedef struct _personstruct
{
  int i_age;
  char i_name[64];

} PERSONINFO;


/* STEP 2: Allocate as many of these instance structures  as the program requires.
      In this case, a constant NUM_PERSONS is used to represent the number of
      structures to allocate.

      Use of a #defined constant is not required.
 */

/* Global data */
#define NUM_PERSONS  20

PERSONINFO person[NUM_PERSONS] = {0};


/* STEP 3:  Create a global pointer to this new instance structure.  This pointer will
later be assigned to point to one of the allocated instance structures.
 */
PERSONINFO *p;


/* STEP 4: Create preprocessor macros that convert references to the original data
variables into references to the new variables within the new instance structure.
 */
#define name   p->i_name
#define age    p->i_age
```

```
/* STEP 5: Create a simple global entity that will handle the memory management for
the different instances. This function or macro only needs to point the instance
pointer ('p' in this example) to the address of one of the allocated structures.
 */
#define SET_PERSON(i)  p = &person[i];


/* Target function - UNCHANGED */
void ShowInfo (void)
{
  printf ("My name is %s.\n", name);
  printf ("My age  is %d.\n", age);
}


/*
 * Main program code.
 *
 * This simple program will repeat the call to ShowInfo()
 * until the program is terminated. During each loop, it
 * will change the data instance to a new instance, so
 * the information for a different 'person' is displayed
 * each time.
 *
 * Although this program does very little useful work, it
 * serves to demonstrate the simplicity and power of the
 * invention. In this case, there is only one function
 * used (ShowInfo) to demonstrate that the original
 * function does not need to change in any way to support
 * multiple instances. Using the invention techniques,
 * there could be ANY number of functions in the program
 * using multiple instances, and none of them would need
 * to change to support multiple instances.
 *
 * There is also no limit to the number of data items and  * data types that can be
included in a program's
 * instance structure. In this case, there are only two
 * data items in the instance structure, but there could
 * be ANY number of different items in a program's
 * instance structure.
 *
 * Only a small amount of work needs to be done to make
 * all of the program's global data work with multiple
 * instances.
 */
main ()
{
  int which_person = 0;

  while (1)
  {
    /* Switch data instances */
    SET_PERSON(which_person);

    /* Target function - UNCHANGED */
    ShowInfo();
```

```
        /* Move on to the next person, wrapping to zero at
          the end of the list. */
        which_person++;

        if (which_person == NUM_PERSONS)
          which_person = 0;
      }
    }
```

Figure 8 is a flow chart of a process for using multi-instanced software to implement a DLC layer of a controllerless modem using DSVD (V.70). A single instance implementation of the LAPM code was processed as discussed in conjunction with Figure 7 to create a multi-instanced version of that code utilizing the preprocessor. That means that maintenance personnel would not have to learn new code. It also means that only one copy of the code would be running. Additional instances of the code would only require data space. The multi-instanced version of the DLC would be opened with multiple instances, passing source priorities as parameters to the DLC module (800). Each source (audio or data in this example) would open a respective DLC instance (i) and receive a handle with which it would refer to that instance (810). The stream with the highest priority (in V.70, audio has priority over data) will be selected and the appropriate instance rendered active (820). Data would be sent from the selected stream (830) and, periodically, a check made to determine whether or not a higher priority set stream request had been received (840). If a higher priority set stream request had been received, the data from the non-selected streams would be queued pending availability (at that level of priority) of the output channel. If the existing source is not preempted by a higher priority set stream request (840-N), more data would be sent until all data from that stream ready for transmission had been sent (860-Y). If there is no more data (860-N), the next highest priority stream having data to transmit will be selected (820) and the process repeats. Thus, single instance software can be utilized intact, using the techniques of this invention and run in a multi-instance fashion with great ease and efficiency.

Figure 9 is a diagram showing a multi-instanced DLC layer showing multiple instances of a line access protocol (LAPM). The description given in conjunction with Figure 8 really extends beyond DSVD (V.70) to an implementation, shown in Figure 9, where any number of instances of the LAPM code can be run in the manner described. The other sources shown in Figure 9 can include facsimile data, image data, text and any of the many data formats that might be sent. Thus, a large number of data sources can be accommodated with only one version of the LAPM code running.

Figure 10 is a diagram showing how multi-instanced software can be utilized to implement a plurality of protocol stacks which can be used simultaneously or

independently in accordance with one embodiment of the invention. As shown in Figure 10, the entire protocol layer 310 can be replicated a number of times or, the protocol stack can be arranged so as to have a single instance of code which can select protocols and thus each instance of the protocol stack can reflect a different protocol. Thus, as shown in Figure 10, by selecting a particular protocol stack instance, different protocols can be run substantially simultaneously with the management of the selection of the particular instance active at a given point in time specified by the global function or macro as discussed at 740.

Figure 11 is a diagram showing how multi-instanced software can be utilized to implement a plurality of modems which can be used simultaneously or independently in accordance with one embodiment of the invention. Entire controllerless software modems (e.g. Figure 3) can be made multi-instanced using the techniques described herein. Such a controllerless modem could contain components, such as the protocol layer, or the DLC layer which are themselves multi-instanced. As a result, one can implement a plurality of software modems as instances of a single code set. As a result, a manufacturer could manufacture and a consumer purchase essentially a plurality of software modems which are instances of a single code set and these could be activated by simply selecting the instance to be utilized at any given point in time. This enables one to effectively have a plurality of modems running substantially simultaneously (or independently), each having different characteristics. In some multiple modem configurations, multiple independent DSP chips may be required, depending on the design, interface, and processing capabilities of each DSP.

There are certain instances in which a processor runs with essentially no operating system. Typically, this might occur in a special purpose device having an embedded processor. By writing an allocation routine (1210) to run on the processor 1200, a plurality of instances of a program structure, having, for example, registers, a program counter, instructions and data can be allocated, each to be selectively activated by the allocation application 1210. In one embodiment, the allocation application can be a simple time slice allocation round robin algorithm which gives each program instance its slice of processor time. Thus, an embedded processor which would otherwise be inherently single instance, can be made multi-instance by effectively implementing memory management in the manner described in Figure 12. This would be particularly advantageous when using a processor which did not have a hardware memory management unit. One could simulate memory management functions even though the processor had no hardware memory management unit.

Figure 13 is an illustration of a technique for enabling multi-tasking on a processor running a single task operating system. Many operating systems, such as

DOS, are essentially single task systems. That is, once an application is opened in DOS, typically no other application can run. By applying the techniques discussed in conjunction with Figure 12 to a process running under DOS, what would otherwise be single instance software which can handle one task can be made multi-instanced (and therefore multi-task capable) and the power and functionality of the DOS operating system expanded beyond that which would otherwise be available.

Although the present invention has been described and illustrated in detail, it is clearly understood that the same is by way of illustration and example only and is not to be taken by way of limitation, the spirit and scope of the present invention being limited only by the terms of the appended claims and their equivalents.

What is claimed is:

1. A method of creating multi-instantiated code from single instance code, comprising the steps of:

a. moving all global data from said single instance code into an instance structure;

b. allocating a plurality of instances of the instance structure;

c. creating a global pointer to selectively point to one of said plurality of instances;

d. converting references to variables in said single instance code to references to variables in an instance structure; and

e. creating a process to control selection of the instance pointed to by said global pointer.

2. The method of claim 1 implemented in a pre-processor.

3. The method of claim 1 in which the step of moving includes pre-pending moved data names with an instance structure prefix.

4. The method of claim 1, in which the step of converting is done in a pre-processor macro.

5. The method of claim 1 in which said process is a function.

6. The method of claim 1 in which said process is a macro.

7. The method of claim 1 in which said process controls priority of activation of an instance.

8. Apparatus for creating multi-instantiated code from single instance code, comprising:

a. a computer, and

b. a compiler configured to convert single instance software into multi-instance software, using said computer.

9. Apparatus of claim 8 in which said compiler includes a pre-processor, said pre-processor converting single instance software into multi-instance software.

10. Computer apparatus, comprising:

a. a computer, and

b. at least one multi-instantiated process, running on said computer.

11. A computer system, comprising:

a. a network; and

b. a plurality of computers connected to said network, at least one of said computers having installed thereon one of (1) a compiler configured to convert single instance software into multi-instance software and (2) multi-instantiated software.

12. A method of creating multi-instantiated code, comprising the step of sending software for converting single instance software into multi-instance software from one computer across a network to another computer.

13. A method of receiving multi-instantiated code, comprising the step of receiving at least a pre-processor for converting single instance software into multi-instance software from another computer across a network.

14. A computer program product comprising:

a. a memory medium; and

b. a computer program stored on said memory medium, said computer program comprising instructions for converting single instance software into multi-instance software.

15. A computer program product comprising:

a. a memory medium; and

b. a computer program stored on said memory medium, said computer program comprising at least one multi-instantiated process.

16. A computer program product comprising:

a. a memory medium; and

b. a computer program stored on said memory medium, said computer program establishing a data structure containing data from a plurality of instances of a software process.

17. A software modem comprising:

a. at least one multi-instantiated code segment; and

b. a communications interface for communicating information processed by said code segment to a communications link.

18. The software modem of claim 14 in which said multi-instantiated code segment includes a protocol stack.

19. The software modem of claim 14 in which said multi-instantiated code segment includes a line access protocol.

20. A modem, comprising:

a. a processor and

b. a plurality of instances of modem software running on said processor which can be activated selectively.

21. The modem of claim 20, in which an active instance is specified by setting a pointer.

22. A computer comprising:

a. a processor and

b. a plurality of instances of modem software running on said processor which can be activated selectively.

23. A software memory management unit, comprising:

a. a data structure containing a plurality of second data structures;

b. a pointer pointing to one of said second data structures; and

c. an allocation process for setting said pointer to select a second data structure.

24. The software memory management unit of claim 23, in which at least one of said second data structures contains information about a program contained within said instance.

25. The software memory management unit of claim 24 in which said information about a program includes at least one of register contents, program counter, instructions and data.

26. A computer comprising:

a. a processor;

b. a single task operating system running on said processor;

c. a data structure containing a plurality of second data structures, each second data structure containing information about a respective software task; and

d. an allocation process for setting said pointer to select a second data structure to enable said respective software task to run under said single task operating system.

27. The computer of claim 22, in which said allocation process selects second data structures on a round robin basis.

28. A method of making a single task operating system into a multi-task operating system, comprising the steps of:

a. creating a data structure containing a plurality of second data structures, each second data structure containing information about a respective software task; and

b. running an allocation process for selecting a second data structure to enable said respective software task to run under said single task operating system.

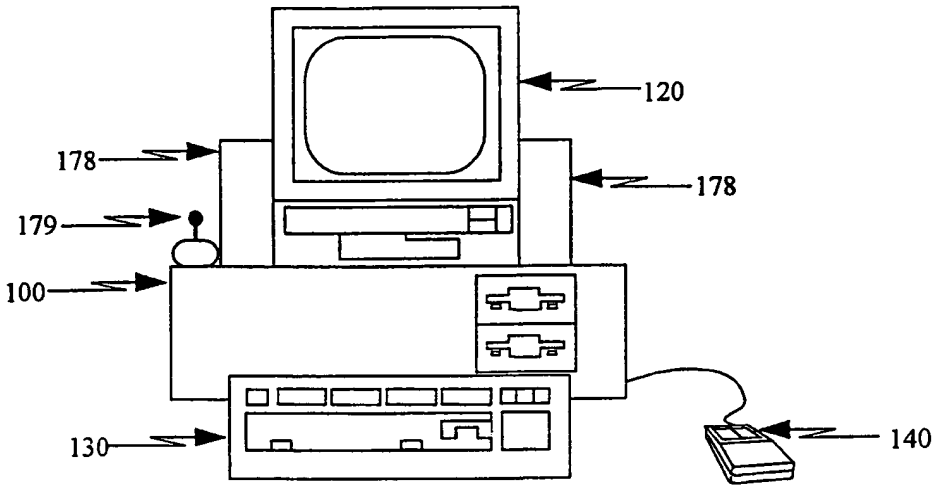


Figure 1A

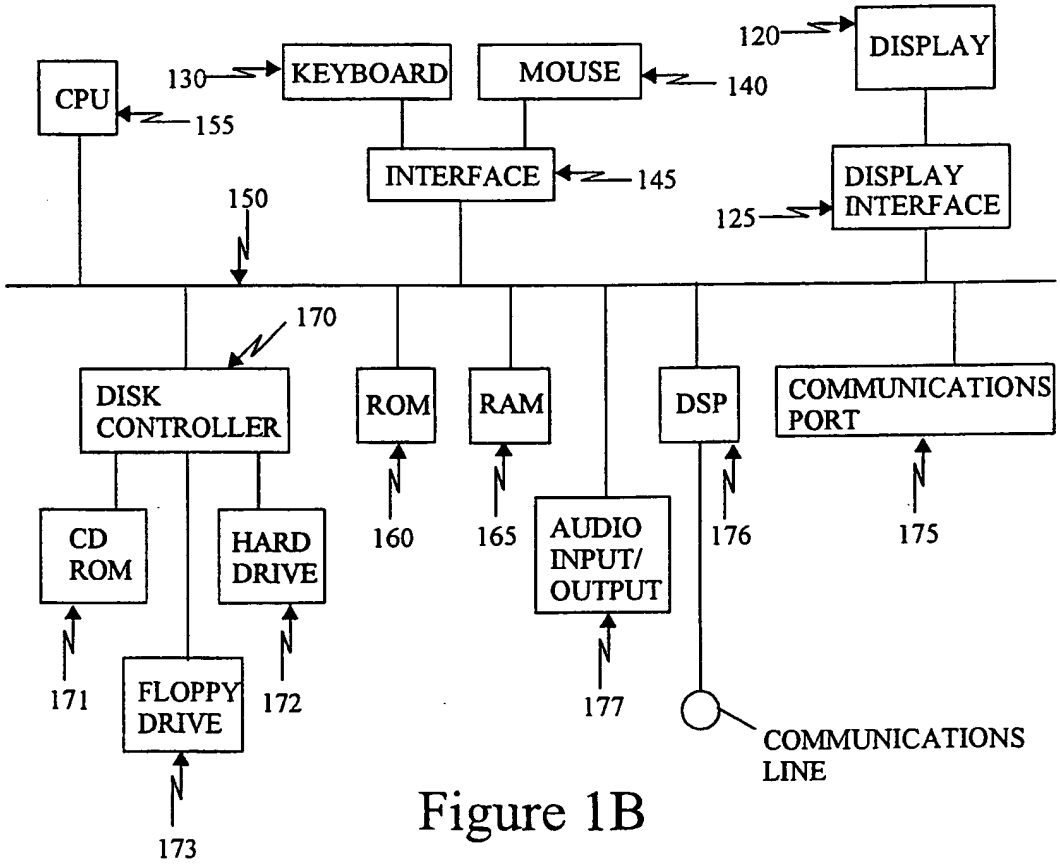


Figure 1B

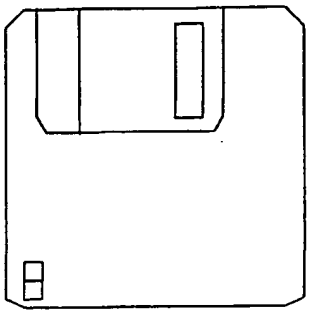


Figure 1C

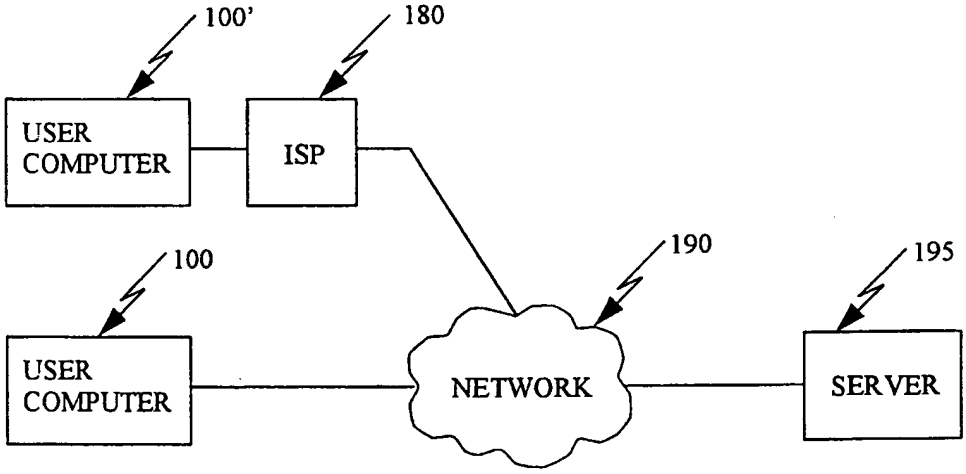


Figure 1D

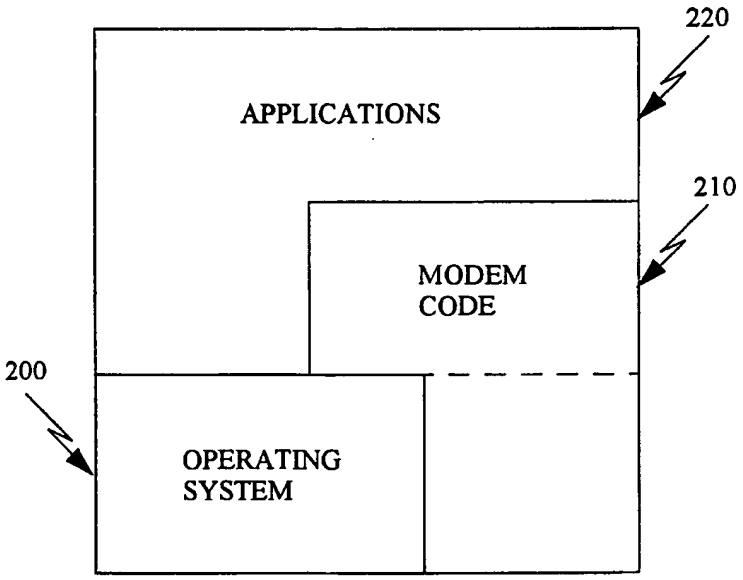


Figure 2

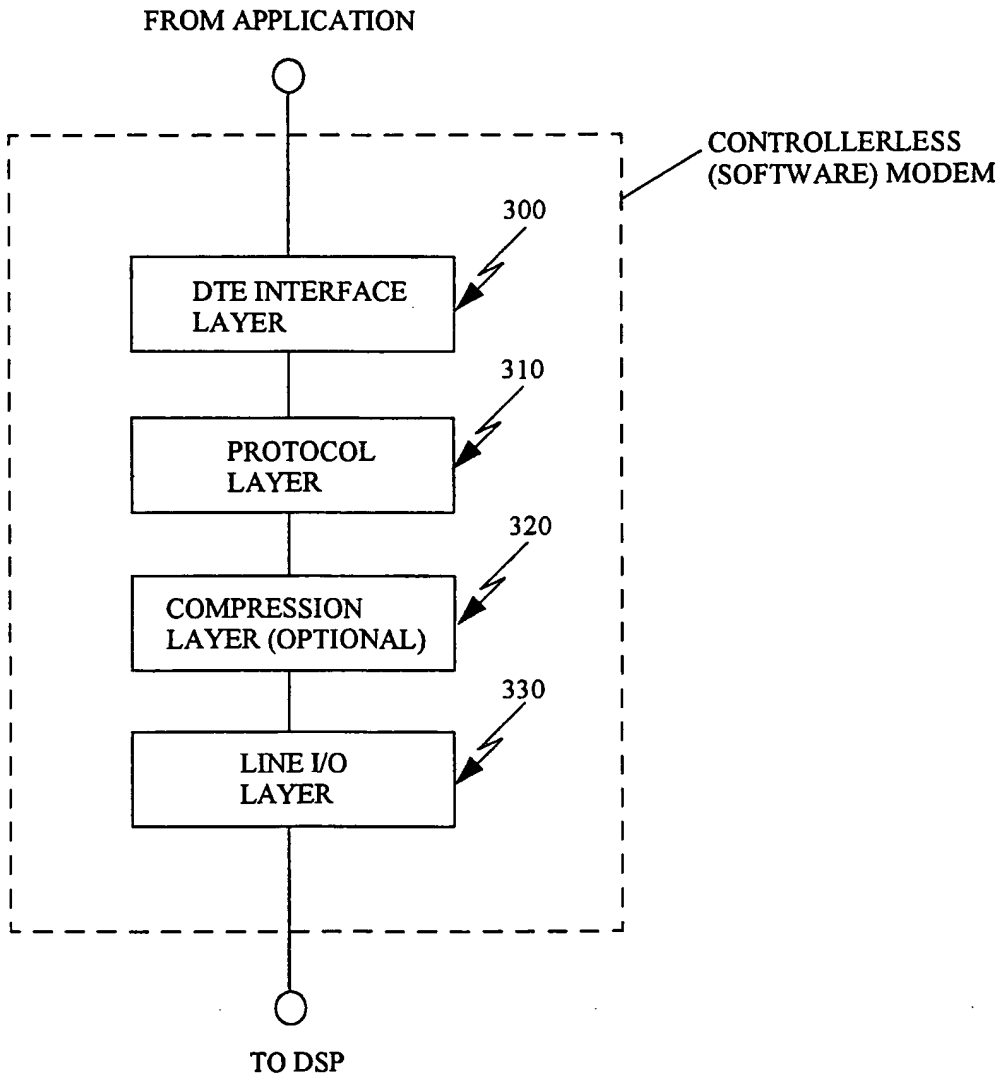

FROM APPLICATION
CONTROLLERLESS
(SOFTWARE) MODEM
300
DTE INTERFACE
LAYER
310
PROTOCOL
LAYER
320
COMPRESSION
LAYER (OPTIONAL)
330
LINE I/O
LAYER
TO DSP

Figure 3

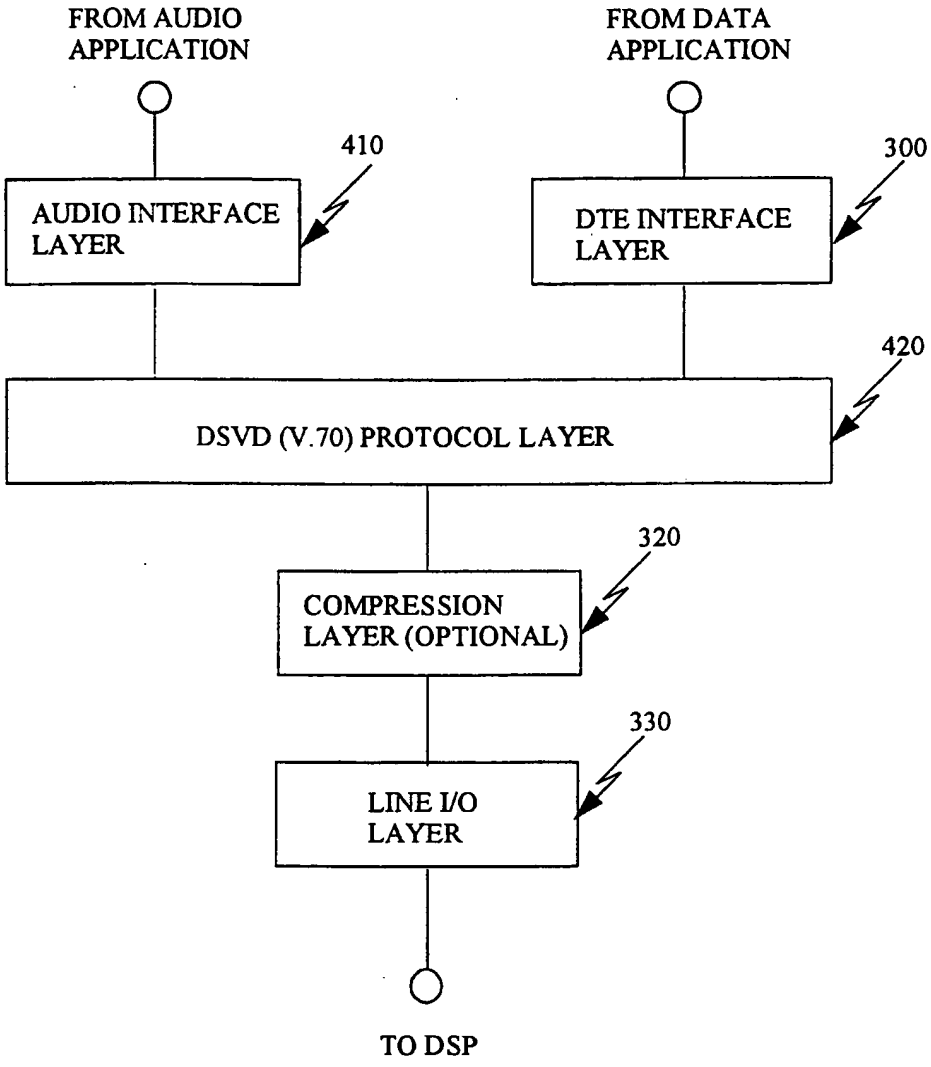

FROM AUDIO APPLICATION
FROM DATA APPLICATION
410
300
AUDIO INTERFACE LAYER
DTE INTERFACE LAYER
420
DSVD (V.70) PROTOCOL LAYER
320
COMPRESSION LAYER (OPTIONAL)
330
LINE I/O LAYER
TO DSP

Figure 4

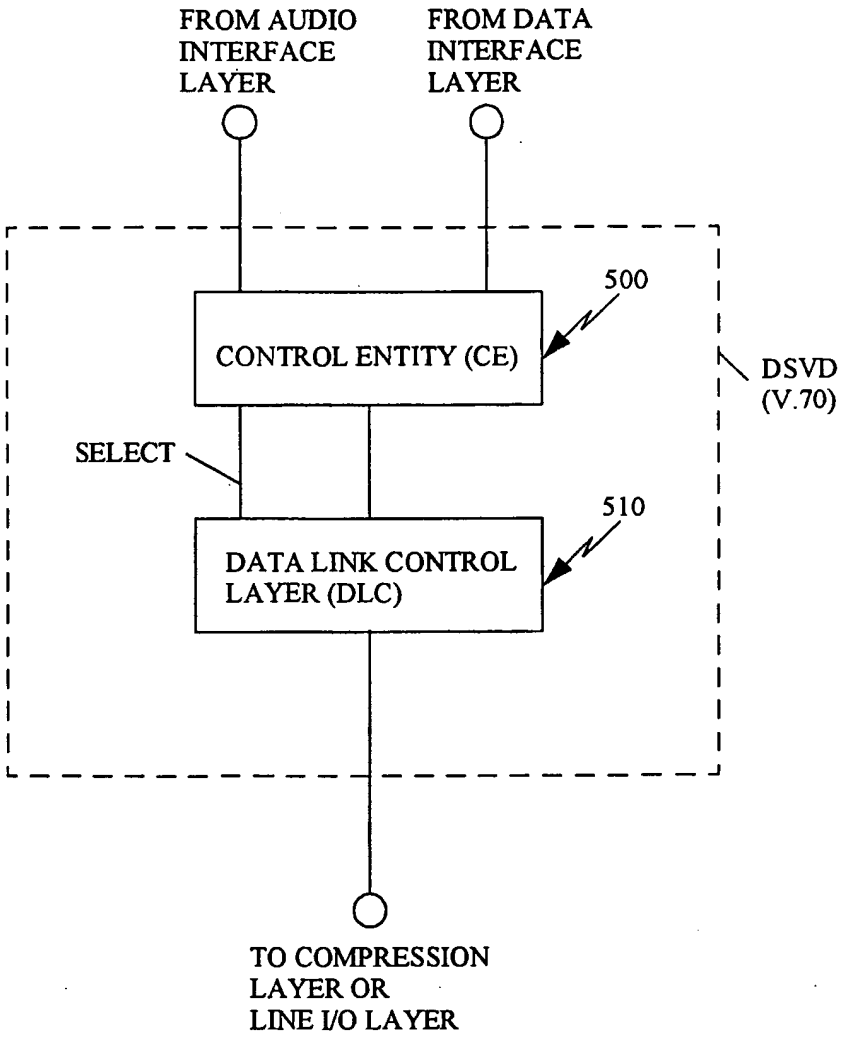

FROM AUDIO INTERFACE LAYER
FROM DATA INTERFACE LAYER
500
CONTROL ENTITY (CE)
DSVD (V.70)
SELECT
510
DATA LINK CONTROL LAYER (DLC)
TO COMPRESSION LAYER OR LINE I/O LAYER

Figure 5

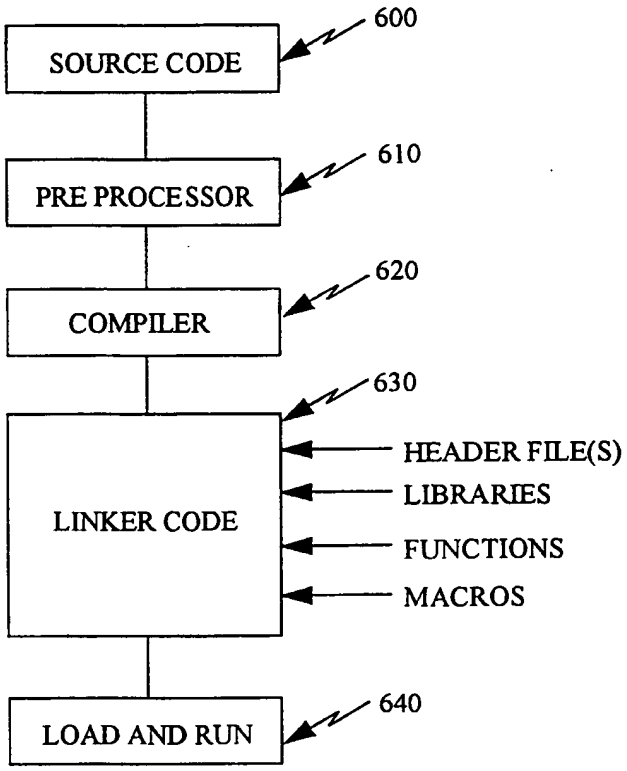

600
SOURCE CODE
610
PRE PROCESSOR
620
COMPILER
630
LINKER CODE
HEADER FILE(S)
LIBRARIES
FUNCTIONS
MACROS
640
LOAD AND RUN

Figure 6

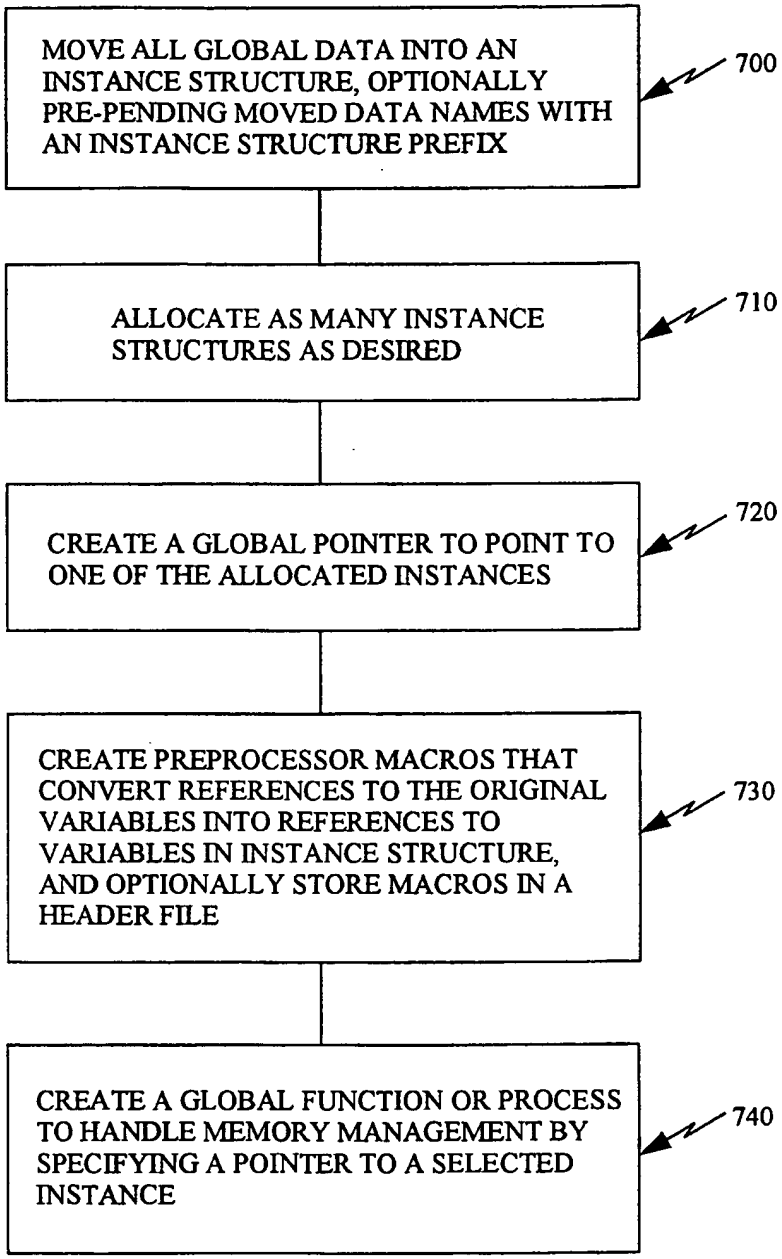

MOVE ALL GLOBAL DATA INTO AN INSTANCE STRUCTURE, OPTIONALLY PRE-PENDING MOVED DATA NAMES WITH AN INSTANCE STRUCTURE PREFIX
700
ALLOCATE AS MANY INSTANCE STRUCTURES AS DESIRED
710
CREATE A GLOBAL POINTER TO POINT TO ONE OF THE ALLOCATED INSTANCES
720
CREATE PREPROCESSOR MACROS THAT CONVERT REFERENCES TO THE ORIGINAL VARIABLES INTO REFERENCES TO VARIABLES IN INSTANCE STRUCTURE, AND OPTIONALLY STORE MACROS IN A HEADER FILE
730
CREATE A GLOBAL FUNCTION OR PROCESS TO HANDLE MEMORY MANAGEMENT BY SPECIFYING A POINTER TO A SELECTED INSTANCE
740

Figure 7

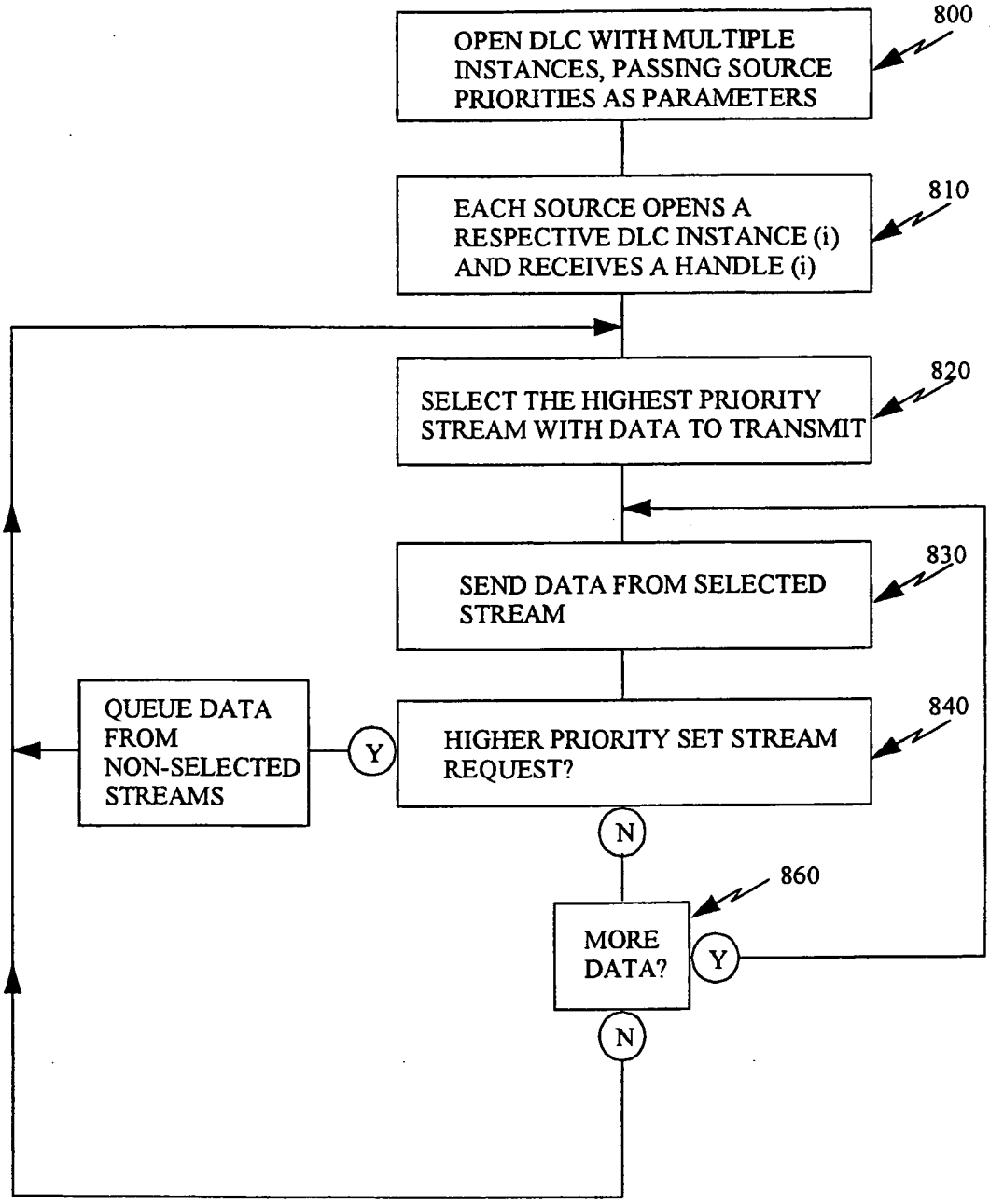

OPEN DLC WITH MULTIPLE INSTANCES, PASSING SOURCE PRIORITIES AS PARAMETERS
800
EACH SOURCE OPENS A RESPECTIVE DLC INSTANCE (i) AND RECEIVES A HANDLE (i)
810
SELECT THE HIGHEST PRIORITY STREAM WITH DATA TO TRANSMIT
820
SEND DATA FROM SELECTED STREAM
830
HIGHER PRIORITY SET STREAM REQUEST?
840
Y
QUEUE DATA FROM NON-SELECTED STREAMS
N
860
MORE DATA?
Y
N

Figure 8

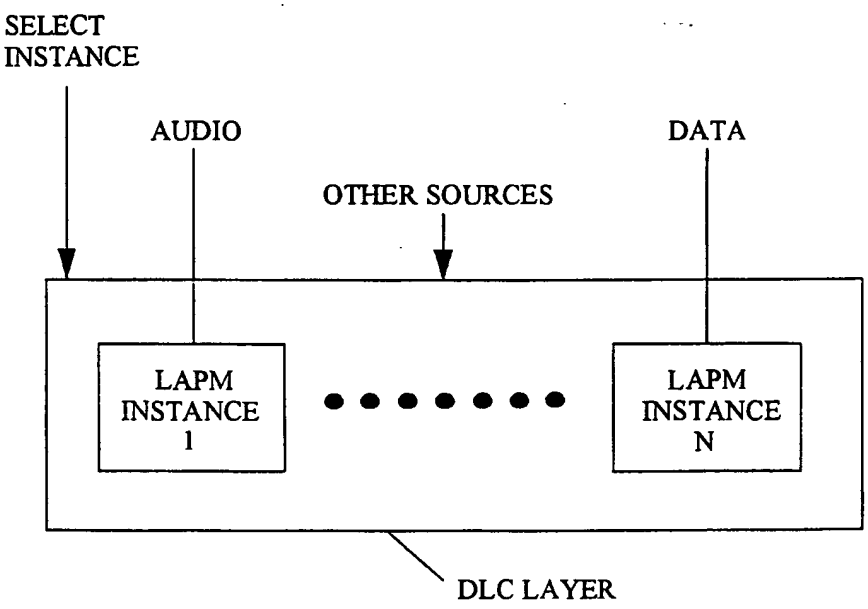

SELECT
INSTANCE
AUDIO
DATA
OTHER SOURCES
LAPM
INSTANCE
1
LAPM
INSTANCE
N
DLC LAYER

Figure 9

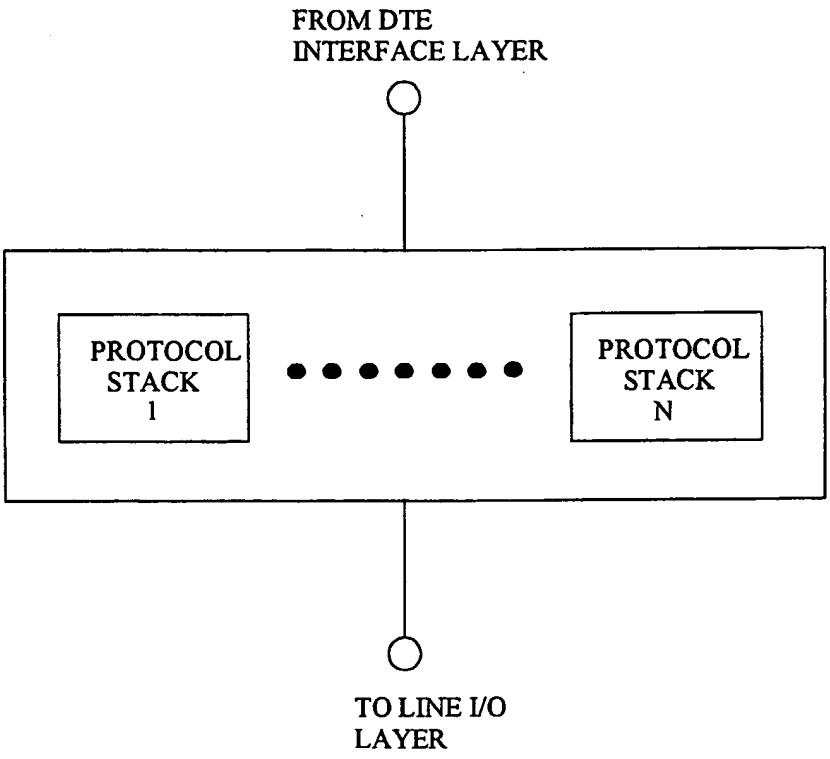

FROM DTE
INTERFACE LAYER
PROTOCOL
STACK
1
PROTOCOL
STACK
N
TO LINE I/O
LAYER

Figure 10

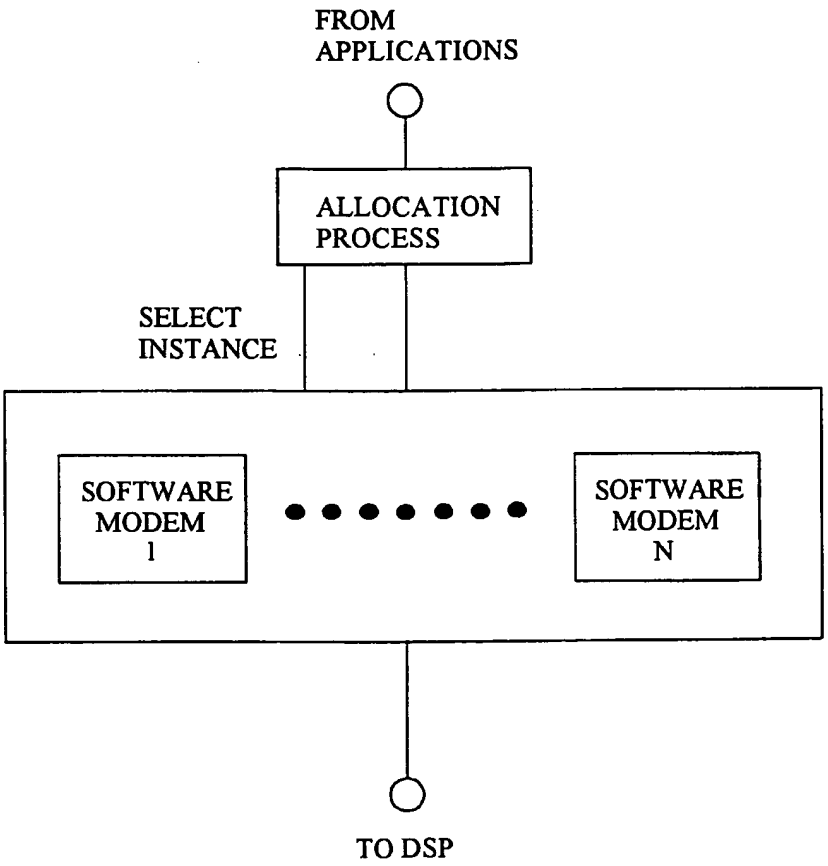

FROM
APPLICATIONS
ALLOCATION
PROCESS
SELECT
INSTANCE
SOFTWARE
MODEM
1
SOFTWARE
MODEM
N
TO DSP

Figure 11

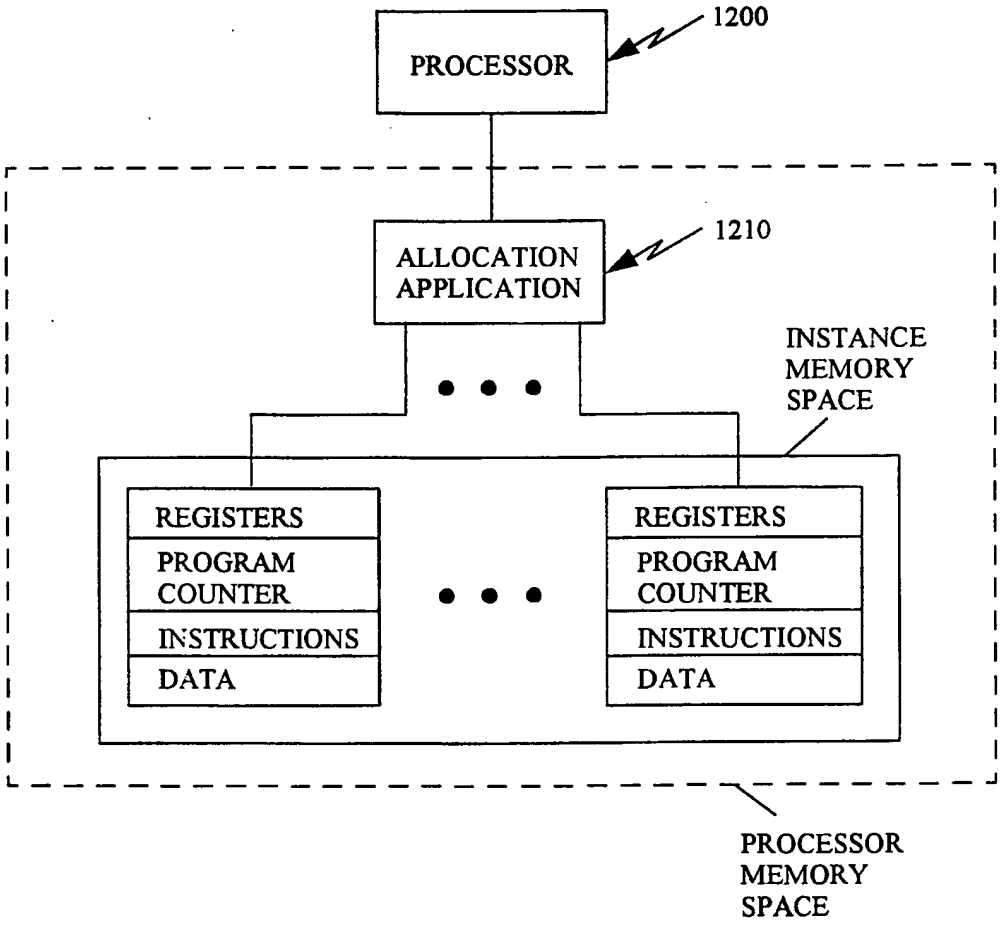

PROCESSOR
1200
ALLOCATION APPLICATION
1210
INSTANCE MEMORY SPACE
REGISTERS
PROGRAM COUNTER
INSTRUCTIONS
DATA
REGISTERS
PROGRAM COUNTER
INSTRUCTIONS
DATA
PROCESSOR MEMORY SPACE

Figure 12

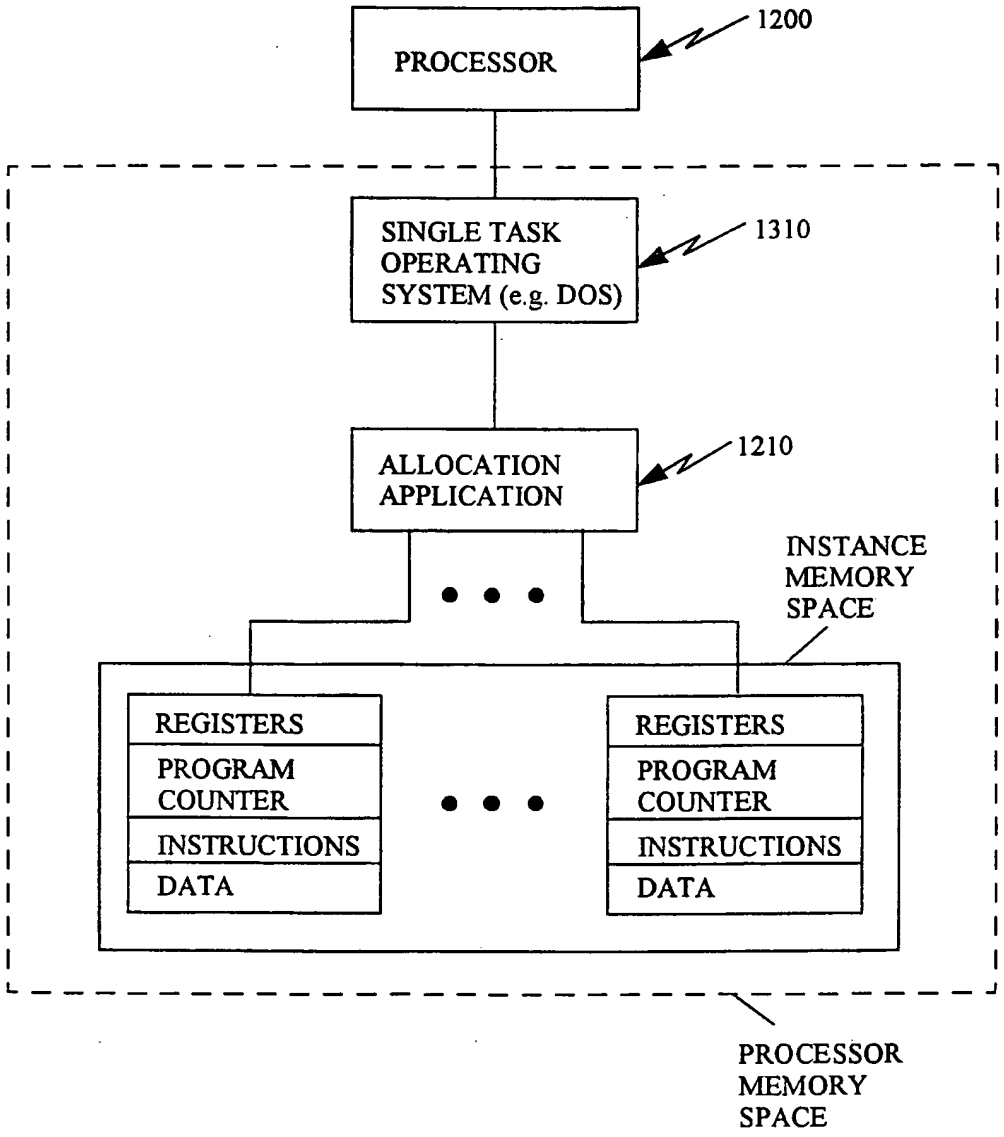

PROCESSOR
1200
SINGLE TASK OPERATING SYSTEM (e.g. DOS)
1310
ALLOCATION APPLICATION
1210
INSTANCE MEMORY SPACE
REGISTERS
PROGRAM COUNTER
INSTRUCTIONS
DATA
REGISTERS
PROGRAM COUNTER
INSTRUCTIONS
DATA
PROCESSOR MEMORY SPACE

Figure 13

INTERNATIONAL SEARCH REPORT

Inte onal Application No
PCT/US 98/15022

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/46 G06F9/40

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 416 768 A (DIGITAL EQUIPMENT CORP) 13 March 1991 see column 2, line 7 - column 6, line 23	1,3,5,6, 8,10, 14-16, 23-28

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
13 November 1998	23/11/1998

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Brandt, J

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int ional Application No
PCT/US 98/15022

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0416768 A	13-03-1991	AT 167582 T	15-07-1998
		DE 69032418 D	23-07-1998
		JP 3142557 A	18-06-1991
		US 5345588 A	06-09-1994